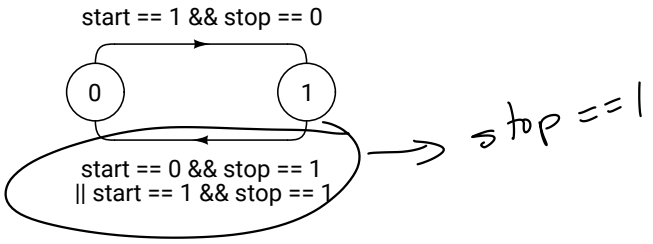
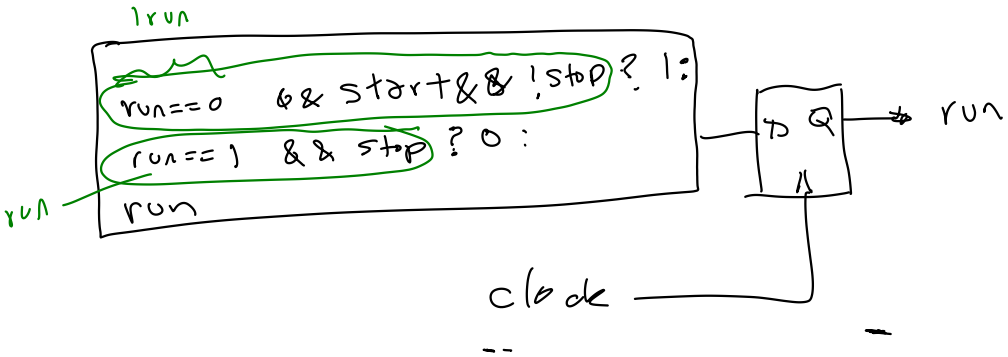
State Machines

Exercise 1:



Draw the schematic for this state machine. Write an **`always_ff`** statement that implements this state machine.




```
always_ff @(posedge clk)
   run <= run==0 && start&& !stop ? 1 :
          run==1 && stop ? 0 :
          run ;
```

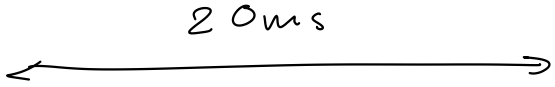
Exercise 2: Simplify the solution using the fact that addition of 1 to 3 "wraps" a 2-bit value to 0. What would you change so the counter does not "wrap around" from 2'b11 to 2'b00?

```
module ex56
   ( output logic [1:0] count,
     input logic reset, clk ) ;

   always_ff @(posedge clk) count
      <= reset ? 2'b00 :
         count == 2'b11 ? 2'b00 :
         count + 1'b1 ;

endmodule
```

remove line if counters "wrap around"

use 2'b11 for count "saturating" at 2'b11.

Exercise 3: What value of N would give a 20 ms delay if the clock frequency is 50 MHz? How many bits are needed for this timer's register?



$$N = \frac{20ms}{20ns} = \frac{10^{-3}}{10^{-9}} = 10^6$$

$f_{clock} = 50MHz$

$$T_{clock} = \frac{1}{f_{clock}} = \frac{1}{50 \times 10^6} = 20ns$$

999,999 → 0

$2^{10} \rightarrow 1024$

$2^{20} \rightarrow (1024)^2 \approx 10^6$.

$2^{30} \approx 10^9$

$2^{40} \approx 10^{12}$

$2^m \geq 999,999$

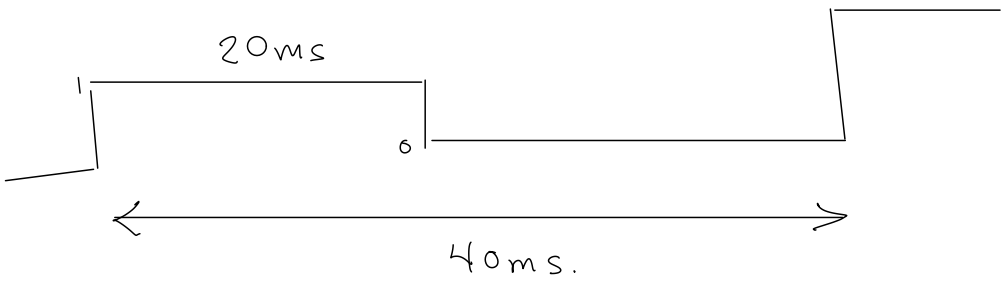
$m \geq \log_2(10^6)$.

19.9 (20)

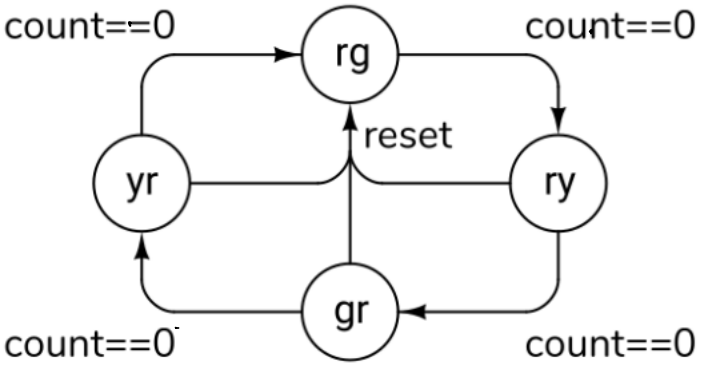
Exercise 4: Assume the timer above is reset to $N - 1$ each time it reaches 0. For how long is the register value 0? What are the period and frequency of a signal that is inverted each time the count reaches 0?

it's 0 for 1 clock period

0, 999,999, ... 1, 0, 999,999, ... 1, 0, 99999|



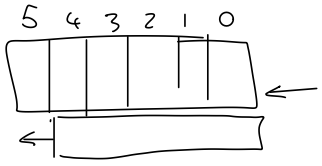
Exercise 5: Write the state transition table for the state machine for the **`lights`** output.

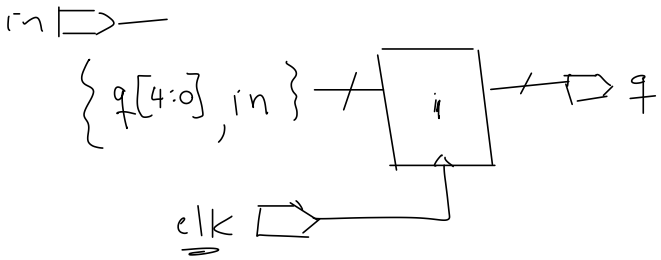


current state (lights)	inputs: reset	inputs: count	next state (next lights)
X	1	X	rg
rg	0	0	ry
ry	0	0	gr
gr	0	0	yr
yr	0	0	rg
S	0	≠0	S

no change

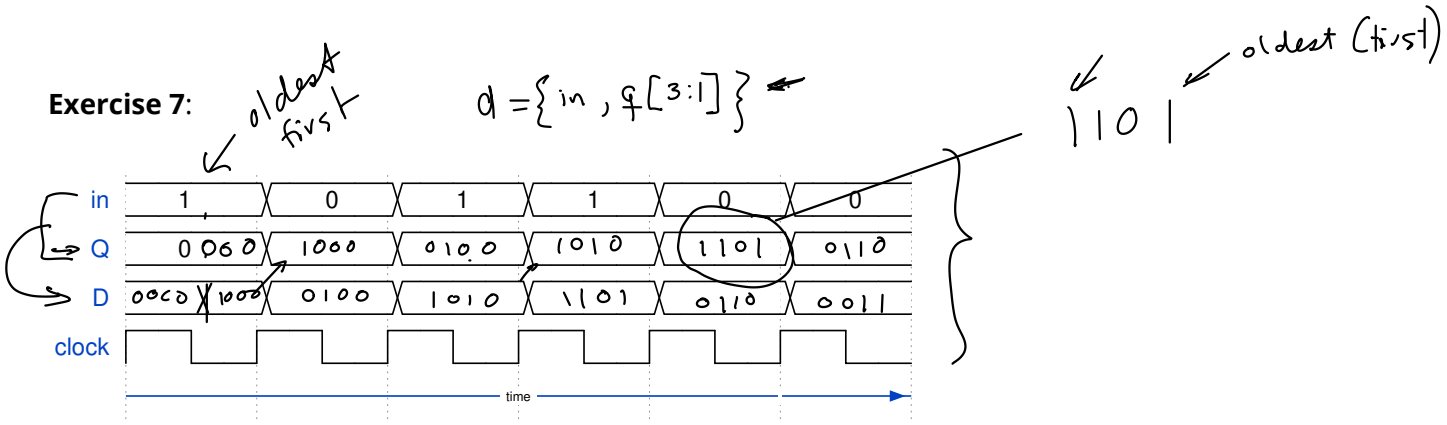
Exercise 6: The example above is an N-bit shift register that shifts the bits right. Draw a block diagram and write the Verilog for a 6-bit shift register that shifts left.





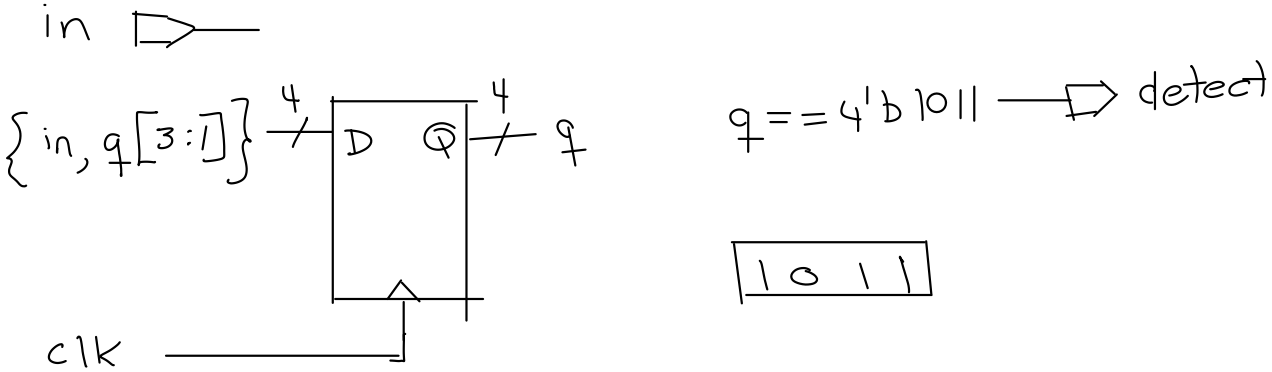

```
module (input logic in, clk,
        output logic [5:0] q );

always_ff @(posedge clk)
  q <= {q[4:0], in};

endmodule
```

Exercise 7:



Fill in the diagram above for a 4-bit ($N = 4$) right-shift shift register. Assume the initial value is zero. Which bit is the oldest (first) value in the Q waveform? Which bit of the shift register holds the oldest value? in

Exercise 8: Draw a block diagram and write the Verilog for a circuit that sets an output named **`detect`** high when the sequence of values 1, 1, 0, 1 has appeared on an input named **`in`** on successive rising edges of the clock.



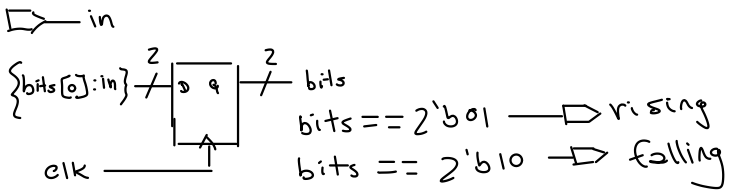

```
module detector (
    input logic in, clk,
    output logic detect );
logic [3:0] q;
always_ff @(posedge clk)
    q <= {in, q[3:1]};
assign detect = q == 4'b1011;
endmodule;
```

Exercise 9: For which states would **`falling`** be asserted? **`rising`**? Draw the schematic and write the Verilog for this state machine. Assume an input **`in`** and a 2-bit register **`bits`** that holds the two most recent input values.

S.R. shifts left

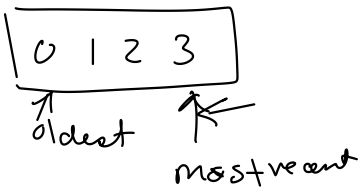
falling asserted for 1 followed by 0 : state = 10

rising for state = 01




```
module edge_detector (
    input logic in, clk,
    output logic rising, falling);

logic [1:0] bits;

always_ff @(posedge clk)
    bits <= {q[0], in};

assign rising = bits == 2'b01;
assign falling = bits == 2'b10;

endmodule
```

Exercise 10: How could you modify the code so that **`digits`** is only updated when an **`enable`** input is asserted?




```
always_ff @(posedge clk) digits
  <= { digits[1:3], digit } ;
```

<= enable ? { digits[1:3], digit } : digits ;

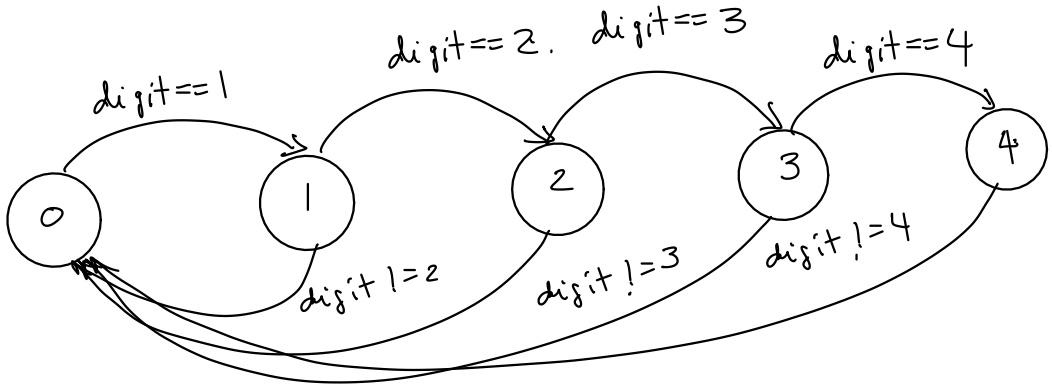
Exercise 11: How many states can this state machine have?

16 bits → 2^{16} possible values (states)

$16 \times 16 \times 16 \times 16 = (2^4)^4 = 2^{16}$

(if constraint digit to 0–9 then $10 \times 10 \times 10 \times 10 = 10^4$ possible states).

Exercise 12: Draw the state transition diagram for this simpler implementation. How many states are there? Write the Verilog using a 3-bit **`count`** state variable.

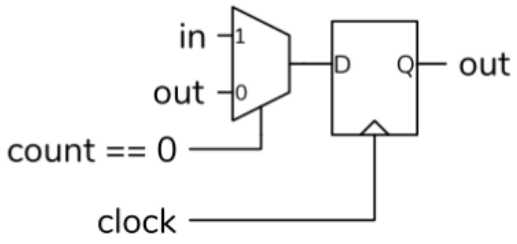


5 states (0–4)

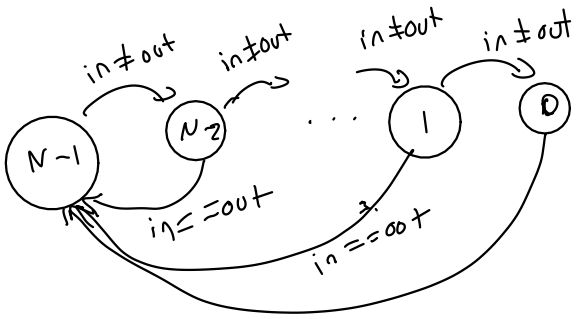
Exercise 13: Write **`always_ff`** statements that implement these two state machines.

count	in == out	next count
x	1	$N-1$
0	x	$N-1$
n	0	$n-1$

or: X 0 count-1



always_ff @(posedge clock)
count <= in == out ? N-1 :
count == 0 ? N-1 :
n - 1;



N-1 0 N-2
N-2 0 N-3
...
4 0 3
3 0 2
2 0 1
1 0 0

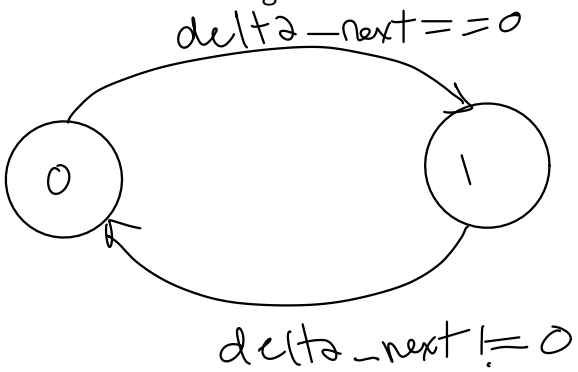
always_ff @(posedge clock)
out <= count == 0 ? in :
out;

Exercise 14: What is the size of the expression `sqrt*sqrt`? Of `{8'b0,sqrt}*sqrt`?

8

16

Exercise 15: Draw the state transition diagram for **done**.



Exercise 16: If we used 8-bits of state information, how many states could be represented? What if we used 8 bits of state but used a "one-hot" encoding?

with 8 bits
- binary encoding: 2^8
- one-hot encoding: 8